2017 IEEE 7th International Advance Computing Conference (IACC)

Fault-Tolerant Scheduling for Scientific Workflows in Cloud Environments

Vinay K[†] and S M Dilip Kumar[†]

[†]*Dept. of Computer Science and Engineering, University Visvesvaraya College of Engineering, Bangalore, India*
ec.vinay@gmail.com, dilipkumarsm@gmail.com

***Abstract*—Executing clustered tasks has proven to be an efficient method to improve the computation of Scientific Workflows (SWf) on clouds. However, clustered tasks has a higher probability of suffering from failures than a single task. Therefore, fault tolerance in cloud computing is extremely essential while running large-scale scientific applications. In this paper, a new heuristic called Cluster based Heterogeneous Earliest Finish Time (C-HEFT) algorithm to enhance the scheduling and fault tolerance mechanism for SWf in highly distributed cloud environments is proposed. To mitigate the failure of clustered tasks, this algorithm uses idle-time of the provisioned resources to resubmit failed clustered tasks for successful execution of SWf. Experimental results show that the proposed algorithm have convincing impact on the SWf executions and also drastically reduce the resource waste compared to existing task replication techniques. A trace based simulation of five real SWf shows that this algorithm is able to sustain unexpected task failures with minimal cost and makespan.**

***Index Terms*—Cloud Computing, Scientific Workflows, Scheduling, Fault tolerance, Task clustering, Failed tasks**

I. Introduction

Scientific workflow applications are composed of many fine computational granularity tasks and merging these tasks into clusters is a common technique used to address execution overheads [1]. However, existing task clustering strategies have ignored the effect of task failures on clouds, despite their significant effect on the large-scale distributed systems such as grids and clouds [2]. The scientists usually require highly distributed systems to compute complex problems that can run for many days or even weeks [3]. If the system is a low fault-tolerant, then it can lose days or even weeks of computation time and it is intolerable for scientists. Therefore, the reliability of a system depends on the fault tolerance mechanisms adopted to recover from a fault that occured before the completion of their applications. The failures also affect the overall SWf execution and increase the makespan. Failures in SWf are mainly due to task failures, workflow level failures and Virtual Machine (VM) failures. Task failure is mainly due to dynamic execution environments, system errors or missing input data. VM failures are basically due to hardware failures. The most prominent fault-tolerant techniques that deals with failures are retry, replication and checkpointing.

Due to increase in the number of scientific applications such as bioinformatics, astronomy, biochemisty, physics, etc. that are migrating to cloud and the ever growing data and complexity of these applications demand a high fault-tolerant computing systems [4]. Most widely used fault tolerance mechanisms are *task resubmission* and *task replication* [5]. In *task resubmission* whenever a task is failed, it is resubmitted either to the same or a different available resource at runtime. Suppose the task is failed due to runtime exception or programming bug, then the resulting resubmission would lead to wastage of valuable resources. Where as *task replication* is a straight forward approach used during the scheduling phase of SWf life cycle. It replicates all the tasks onto available resources. If one task fails, another replicated task will balance the workflow execution. This approach assures high level of fault-tolerance, if there are enough resources available [3].

In this work, we propose a new heuristic algorithm to improve the fault-tolerance of SWf on cloud. The SWf are usually represented as Directed Acyclic Graphs (DAGs) having control-flow and data-flow dependencies. A Heterogeneous Earliest Finish Time (HEFT) heuristic is integrated with task clustering to manage failed tasks during execution. HEFT [6] has the advantages of easily realizing and quickly converging so that the fault-tolerant based scheduling approach is able to get optimal solution in a shorter computational time. The proposed C-HEFT algorithm is extended using standard HEFT algorithm to produce efficient cluster based task scheduling and mapping of heterogeneous resources.

The rest of the paper is organized as follows. Section II provides an overview of the related work with respect to fault-tolerant strategies imposed on scientific applications. Problem formulation is presented in Section III. The proposed mechanism and evaluation of our methodology is presented in Section IV. The simulation results, conclusion, and future enhancement are presented in Section V and VI respectively.

II. Related Work

The existence of faults are often unpredictable in distributed computing systems. Limited works have been appeared in the literature since the inception of cloud computing that aims fault-tolerant based scheduling of SWf [7]. There are mainly two fundamental recognized techniques that support dynamic fault-tolerant based scheduling in distributed systems: resubmission and replication. Resubmission is concerned with resubmitting the tasks to the system again during fault that occurred in the resource on which the task was allocated. And also resubmission may lead to much late finish time

978-1-5090-1560-3/17 $31.00 © 2017 IEEE
DOI 10.1109/IACC.2017.35

for tasks and fail to meet deadlines. On the other side, the task replication creates multiple copies of a task and assign each copy to different resources to ensure successful execution of the task before its deadline [8]. Xiaomin Zhu et al. [9] developed a fault-tolerant model that extends the traditional Primary Backup model on cloud and further, proposed a dynamic fault-tolerant scheduling algorithm to improve the resource utilization and execution of SWf tasks in the presence of node failures in virtualized clouds. Weiwei Chen et al. [1] proposed a general task failure model and three fault-tolerant clustering strategies to increase the runtime performance of SWf executions in faulty environments. Rodrigo N. Calheiros et al. [10] proposed an algorithm that uses idle-time of provisioned resources to replicate tasks and meet its deadlines. A dynamic task scheduling algorithm for Heterogeneous called a Clustering Based HEFT with Duplication (CBHD) is proposed in [11] to improve the load balancing in the heterogeneous environment. A fault-tolerant elastic scheduling algorithms for real-time tasks in clouds named FESTAL, that aimed for both fault tolerance and high resource utilization in clouds is proposed in [12]. A new heuristic called resubmission impact to handle the faults during the execution of SWf tasks in distributed systems is proposed in [3]. Most of the related approaches are based on the predictions of failure probability of a task on a resource in a certain time interval and also budget surpluses due to replication of tasks. In contrast to the previous work, our approach is to group tasks into clusters and schedule them onto available resources and dynamically resubmit the failed tasks from the cluster onto the available idle resources resulting efficient resource utilization and successful execution of the SWf tasks in faulty distributed systems.

III. Problem Formalization

In this section, we enumerate application and resource models of SWf, and formalize the problem of fault-tolerant based scheduling with clustered tasks resubmission on cloud.

A. Application and Resource Models

The application considered is SWf modelled as DAGs and is given by a structured graph $G_w = (T_w, E_w)$, where T_w= $\{t_1,t_2,t_3,..,t_p\}$ is a set of precedence constrained tasks. There exists edges E_w, such that e_{j_k}= (t_j,t_k) is a dependency edge between tasks t_j and t_k, in which t_j is said to be parent of t_k and t_k is the child of t_j. Suppose the parent tasks failed during execution, then a child task have to wait until all its parent tasks are re-executed. Each task t_i is executed by determining its parent tasks, more accurately the one that completes the communication at the latest time. The earliest start time (EST) and earliest finish time (EFT) of a task t_i respectively are defined as:

$$EST(t_i) = \begin{cases} 0, & \text{if } t_i = t_0 \\ max_{t_p \in P_i} EST(t_p) + e_p, & \text{otherwise.} \end{cases} \quad (1)$$

$$EFT(t_i) = EST(t_i) + e_i \quad (2)$$

where t_0 is a entry task without any predecessors, t_p is the parent task of t_i, P_i is the set of parent tasks of t_i and e_p and e_i are the execution time of t_p and t_i respectively. We assume a resource model alike to Amazon's EC2, where VMs are leased on-demand and are charged on hourly basis. In this work, we have considered heterogeneous Virtual Machines (VMs) of different specifications to deal with failed tasks during runtime. We model a VM type, in terms of its processing capacity, and incurs a different cost per usage.

B. Problem Statement

The problem we address in this work is to find an efficient mapping of SWf tasks onto heterogeneous VMs, such that the schedule is fault-tolerant due to uncertainties in the system, and the makespan and cost is minimized. Our solution focuses on dynamic workloads, more specifically dependent tasks, each of which can be large-scale scientific simulations that are common in scientific applications. Suppose that the VM set VM= $\{vm_1, vm_2, ..., vm_m\}$ is a set of heterogeneous VMs and W= $\{w_1, w_2, ..., w_n\}$ is a set of SWf and each SWf w_i= $\{t_1, t_2, ..., t_k\}$ has a set of dependent tasks. The problem is to efficiently execute these dependent tasks on corresponding heterogeneous VMs in a faulty distributed execution environment. The tasks in the SWf are merged into clusters called clustered jobs at different horizontal levels in order to execute clustered jobs in parallel.

For instance, if the tasks t_1, t_2, and t_3 are on the same horizontal level of SWf, then these multiple tasks are merged into single cluster, say k_1 for execution. Similarly, the tasks which are on the different horizontal levels are merged into different clusters such as k_2, k_3 and so on before execution. If any of the tasks failed during execution, rather than executing the entire SWf again, a cluster where the failed task belongs is re-clustered again with only failed tasks and re-executed on the different available idle VMs in order to reduce the overall makespan and cost of SWf.

IV. Proposed Model

This section describes the system architecture of fault-tolerant Scientific Workflow Management System (SWfMS) and detailed explanation of fault-tolerant based scheduling algorithms for SWf in faulty cloud environments.

We propose a fault-tolerant architecture of SWfMS on cloud as shown in Fig. 1 to manage failed tasks of SWf. This architecture fits several SWfMS such as Askalon[1], Pegasus[2], and Taverna[3]. It consists of four major components, namely, *workflow-mapper*, *workflow-engine*, *job-scheduler* and *failure-monitor*. In this work, we have considered a single execution site which consists of multiple VMs. The SWf clustered tasks are executed remotely on separate worker nodes. The *workflow-mapper* generates an executable-workflow from an abstract-workflow [13] (DAG files and other metadeta information) provided by the SWf user . It creates a list of tasks

[1]http://www.askalon.org/
[2]https://pegasus.isi.edu/
[3]http://www.taverna.org.uk/

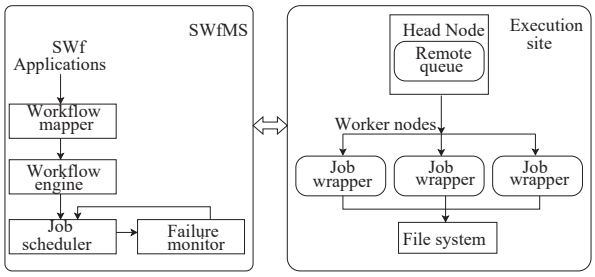


Fig. 1: System Architecture of fault-tolerant SWfMS

TABLE I: Computation time of tasks on different VMs

Task	VM_1	VM_2	VM_3	**Average**	$rank_u(t_i)$
1	14	16	9	13.00	108.00
2	13	19	18	16.66	77.00
3	11	13	19	14.33	80.00
4	13	8	17	12.66	80.00
5	12	13	10	11.66	69.00
6	13	16	9	12.66	63.33
7	7	15	11	11.00	42.66
8	5	11	14	10.00	35.66
9	18	12	20	43.33	44.33
10	21	7	16	14.66	14.66

which has to be submitted to an execution site and also merges tasks into a single-clustered job and later a job is considered as a single execution unit in the SWfMS. The *workflow-engine* executes the single-clustered job, if its parent jobs have completed their execution. The *workflow-engine* depends on the resources such as compute, memory and storage. The time between the single-clustered job release from *workflow-engine* and submission to the *job-scheduler* is denoted as *workflow-engine delay*. The *job-scheduler* manage individual clustered jobs and execution on remote resources. *Failure-monitor* gathers the information such as resource id, failed task id and job id of clustered jobs which failed during execution, and these information are provided to the *job-scheduler* for resubmission. The *job-wrapper* in the execution site extracts tasks form clustered jobs and executes it on the worker nodes.

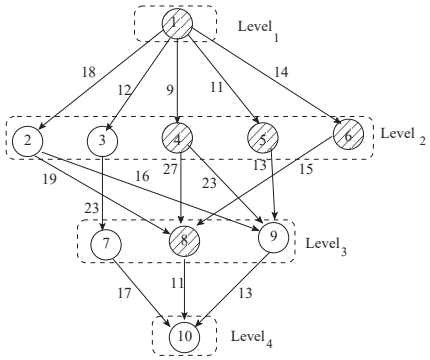


Fig. 2: Example of SWf

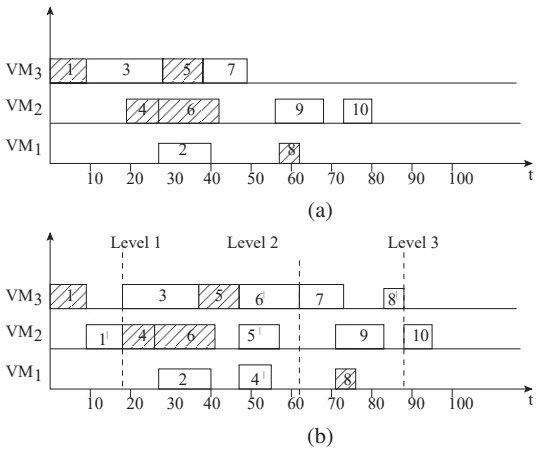


Fig. 3: Schedule of the SWf from Fig. 2. (a) Original scheduling enabled by the HEFT algorithm [6]. (b) Utilization of an available idle slot of VMs for execution of failed tasks.

Among most of the scheduling algorithms for heterogeneous system, the HEFT algorithm has produced shorter schedule lengths of SWf. In this work, we address the tasks failure during execution and propose a C-HEFT algorithm which extends HEFT algorithm for scheduling and the horizontal clustering is used to merge multiple tasks within the same horizontal level of the SWf. If any of the task(s) fails during execution, rather than executing the entire SWf again, the failed task(s) of that particular horizontal cluster is re-clustered which consists of only failed tasks and they failed are resubmitted to idle slot of VMs for execution. Fig. 2 shows the example of SWf which consists of 10 tasks, having dependency between each tasks and the number shown above arrows indicates the communication time between the tasks. The tasks are horizontally clustered for execution. In this example, the tasks $1, 4, 5, 6$ and 8 highlighted with dashed line indicates that the tasks are failed during execution and these failed tasks are re-clustered again at their respective horizontal cluster and resubmitted to the idle slot of VMs for execution with a minimum increase in makespan and cost as shown in Fig. (3a) and Fig. (3b). The SWf shown in Fig. 2 has 50% failure rate, the standard HEFT scheduling algorithm as shown in Fig. (3a) spans 80 seconds with 5 tasks under failure to complete SWf. The schedule shown in Fig. (3b) spans 95 seconds with zero task failure.

We assume that the computing environment consists of a set R of r heterogeneous VMs. Additionally, execution of tasks for a given SWf are assumed to be non-preemptive. P is a m x n computation cost matrix in which each $q_{i,j}$ gives the execution cost of task t_i on VM vm_j. The average execution cost of a task t_i is defined in Equation (3). After calculating the average execution cost of tasks, DAG of a SWf is traversed upwards and a rank value is assigned to each task. The rank value is calculated using the Equation (4) and the values are tabulated as shown in Table I, which is a summation of maximum value resulting from all possible successor

tasks. According to rank value, the tasks are scheduled onto heterogeneous VMs.

$$\bar{q_i} = \frac{\sum_{j=1}^{n} q_{i,j}}{n} \bar{c} \quad (3)$$

$$rank_u(t_i) = \bar{q_i} + max_{t_j \in s_{(t_i)}} \left(\bar{c}_{i,j} + rank_u(t_j) \right) \quad (4)$$

where $\bar{q_i}$ is the average computation cost of task t_i, $s_{(t_i)}$ is the set of immediate successors of task t_i and $\bar{c}_{i,j}$ is the average computation cost of edge $e_{i,j}$. When tasks t_i and t_j are executed on the same VM, the computation cost is zero. Algorithm 1 presents the pseudocode of cluster based

Algorithm 1 C-HEFT Scheduling algorithm

Input: W: Workflow; C: Maximum number of tasks per cluster; T: Number of tasks per workflow; VM_m: Number of VMs

1: **procedure** *RANKING (T)*
2: **for** (i=1 : $number\_of\_tasks$) **do**
3: Compute $rank_u$ of tasks using Equation (4);
4: **end for**
5: $SL \leftarrow \{\}$;
6: $SL \leftarrow sort(rank_u)$;
7: **end procedure**
8: **procedure** $TASK\_CLUSTERING$ *(W,SL,C)*
9: **for** (i=1; $level_i$<$depth(W)$; i++) **do**
10: $C_i \leftarrow$ merge tasks at $level_i$;
11: **end for**
12: $CL \leftarrow$ (C_i,SL);
13: **end procedure**
14: **procedure** *MAPPING (CL,*VM_m*)*
15: **while** CL is unscheduled **do**
16: SELECT C_i from CL
17: **for** (i=1; VM_i<VM_m; i++) **do**
18: Compute EST(C_i, VM_i);
19: Assign C_i to VM VM_j that minimized
20: EFT of cluster;
21: **end for**
22: **end while**

scheduling for given SWf. The notations used in the algorithms are listed in Table II. The *RANKING* procedure computes the computation cost of each tasks on heterogeneous VMs and prepares a Scheduling List (SL) in upward ranking order. The *TASK_CLUSTERING* and *MAPPING* procedures are used to merge tasks at different horizontal levels and mapped onto different heterogeneous VMs for execution respectively. Algorithm 2 shows the pseudocode of the *RECLUSTERING* for the failed tasks. The tasks which are failed in the first attempt are merged into a new cluster at different horizontal levels for execution. And also this approach is simple to incorporate into existing SWfMS with minimum impact on the SWf execution efficiency. Algorithm 3 shows the pseudocode for the *MAPPING* procedure of failed tasks that are resubmitted to the idle VMs.

TABLE II: The Notations used in the Algorithm Design

Parameters	Definitions
SL	Scheduling List
$level_i$	Level of the workflow
C_i	Cluster identifier
CL	Cluster list
VM_m	Total number of VMs
VM_i	Idle VM
C_{new}	New cluster for failed tasks
CL_{new}	New cluster list

Algorithm 2 Task Reclustering algorithm

Input: W: Workflow; C: Maximum number of tasks per cluster

1: **procedure** *RECLUSTERING (CL)*
2: **for** all CL in W **do**
3: **if** task t is failed in C_i **then**
4: $C_{new}.add(t)$;
5: **end if**
6: **end for**
7: $CL_{new} \leftarrow CL$ + C_{new};
8: **end procedure**

V. Experimental Evaluation

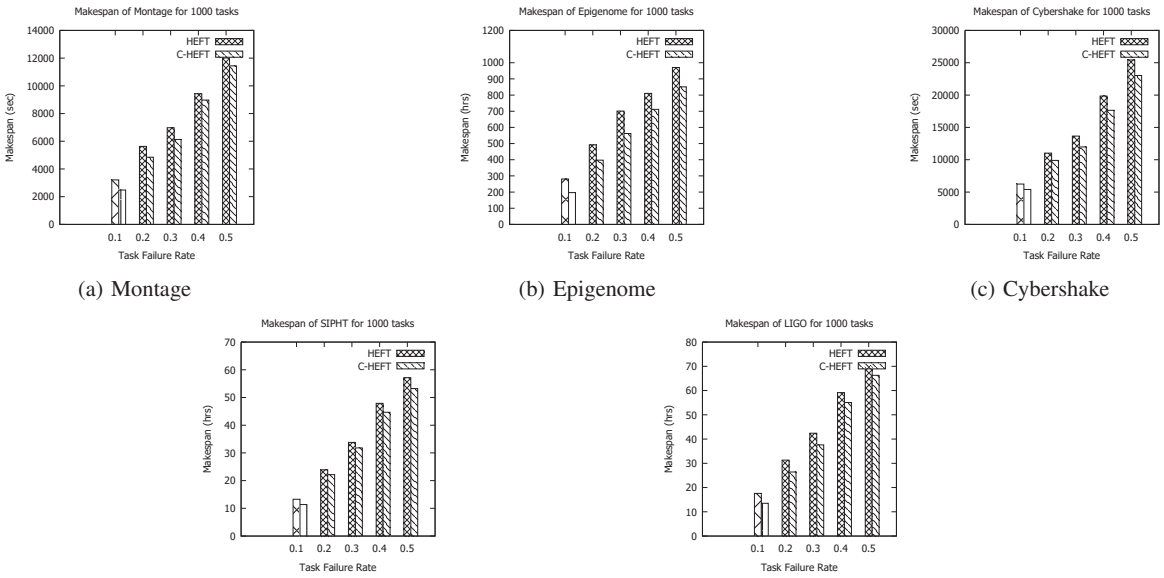
In this section, the experiments conducted to evaluate the proposed C-HEFT, task clustering and resubmission algorithms. Experiments were conducted using CloudSim toolkit [14]. The simulation testbed has a datacenter containing 10 heterogeneous VMs. The datacenter models Amazons EC2 standard instance types, and the parameters relevant for the experiments are presented in Table III. We have assumed there are no VM provisioning delays and the billing period is 60 minutes. Five SWf applications were used in the tests.

Algorithm 3 Resubmission algorithm

Input: W: Workflow; C: Maximum number of tasks per cluster

1: **procedure** *MAPPING(*CL_{new}*,*VM_m*)*
2: **while** CL_{new} is unscheduled **do**
3: SELECT random idle VM VM_i for cluster CL_{new};
4: **for** (i=1; VM_i<VM_m; i++) **do**
5: Compute EST(CL_{new}, VM_i);
6: Assign CL_{new} to VM VM_j that minimized
7: EFT of cluster;
8: **end for**
9: **end while**
10: **end procedure**

They are Montage (production of sky mosaics), Epigenome (analyze human epigenomic data), Cybershake (earthquake risk characterization), SIPHT (bioinformatics) and LIGO (detection of gravitational waves). These SWf applications has different workflow pattern, different data and computational characteristics, and are characterized by Juve et al. [4]. However, experiments were carried out with four different task sizes $(50, 100, 500, 1000)$, we only present results obtained



(a) Montage (b) Epigenome (c) Cybershake

(d) SIPHT (e) LIGO

Fig. 4: Makespan of SWf with respect to Task Failure Rate

TABLE III: VM types used in Experiments

Type	Memory(GB)	Cores	Cost($)
m1.small	1.7	1	0.05
m1.medium	3.75	1	0.1
m1.large	7.5	2	0.2

for 1000 tasks due to the similarity of results, and the inter-arrival time of task failure are varied to fully explore the performance of our fault-tolerant based scheduling algorithm. The observed output metrics are makespan, cost incurred during the execution of SWf and the distribution of workload on heterogeneous resources.

A. Results and Analysis

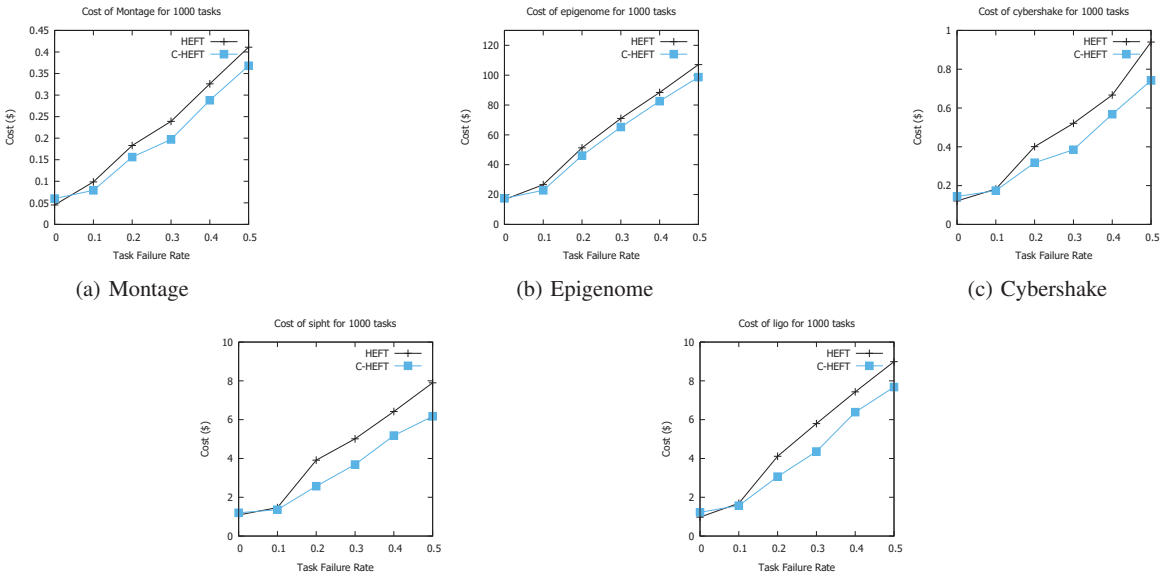
1) Makespan Evaluation: The makespan obtained for different SWf are depicted in Fig. 4. The reference bars in the makespan plot corresponds to 1000 tasks. From the graph, it is clearly seen that the makespan increases with the increase in task failure rate in both the HEFT and C-HEFT algorithms. The proposed algorithm considers idle-time of the VMs to schedule failed tasks for re-execution, thus consumes less time to complete SWf as compared with HEFT algorithm.

2) Cost Evaluation: Fig. 5 shows the total cost of computation for different SWf. The cost includes the failed tasks that are re-executed along with the successful execution of tasks. The cost obtained by C-HEFT based task-resource mapping increases much slower than the heuristic HEFT algorithm. Cost of both the algorithms increases with the increase in task failure rate. Since the failed tasks are re-executed that will incur additional execution cost on the total execution cost of the SWf. The C-HEFT algorithm utilizes idle-time of the VMs to allocate failed tasks in order to reduce total execution cost of the SWf.

3) Distribution of SWf tasks Evaluation: The distribution of different SWf tasks onto available VMs are depicted in Fig. 6. The x-axis represents different SWf consisting of 1000 tasks and y-axis represents the average number of tasks executed by a different compute resource. This evaluation is important as the algorithm chooses different resources to submit tasks. The algorithm restricts all tasks being mapped onto the same resource, so that the tasks can be executed in parallel to increase the efficiency of SWf with a minimum cost. In Fig. 6, Montage and Cybershake SWf uses *m1.small* resource to about 50%, since the total execution time of these workflows are less as compared with other workflows. Whereas Epigenome, LIGO workflow uses *m1.large* more than 40% due to high execution time. Our heuristic approach minimizes the total execution cost and balance the load onto available resources.

VI. Conclusion and Future work

In this paper, we proposed a new heuristic called C-HEFT to improve the fault tolerance of clustered tasks and applied them to five widely used SWf. Experimental results showed that the proposed method significantly reduce the makespan and cost of SWf when compared with an HEFT heuristic algorithm. The idle-time of VMs are considered to save computation cost of failed tasks for re-execution. This work focuses on the evaluation of the fault-tolerant on heterogeneous resources. To study the performance of our proposed work, the failure rate of tasks are varied and from the results it is known that higher the failure rate, more will be the makespan and cost of SWf. Compared to related work, the proposed approach is straight forward and handles failures during runtime. The overall



(a) Montage (b) Epigenome (c) Cybershake

(d) SIPHT (e) LIGO

Fig. 5: Cost of SWf with respect to Task Failure Rate

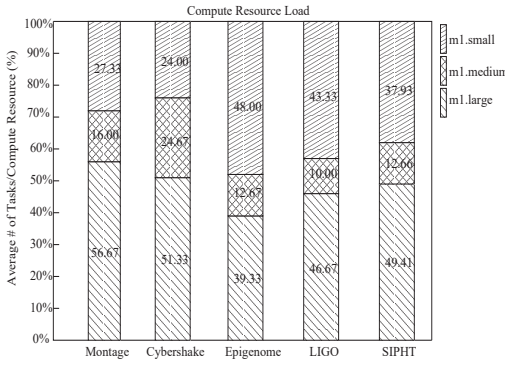


Fig. 6: Distribution of SWf tasks on three different VMs

success rate of SWf tasks are high and easy to implement on commercial cloud systems for smooth execution of SWf. In future, we plan to introduce a task failure model, study the performance of SWf with more accuracy in an unstable environments and propose workload and fault prediction models.

REFERENCES

[1] Weiwei Chen, Rafael Ferreira da Silva, Ewa Deelman, and Thomas Fahringer. Dynamic and fault-tolerant clustering for scientific workflows. *IEEE Transactions on Cloud Computing*, 4(1):49–62, 2016.

[2] John Bresnahan, Tim Freeman, David LaBissoniere, and Kate Keahey. Managing appliance launches in infrastructure clouds. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, page 12. ACM, 2011.

[3] Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. A new fault tolerance heuristic for scientific workflows in highly distributed environments based on resubmission impact. In *e-Science, 2009. e-Science'09. Fifth IEEE International Conference on*, pages 313–320. IEEE, 2009.

[4] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[5] Kassian Plankensteiner, Radu Prodan, Thomas Fahringer, Attila Kertesz, and Peter K Kacsuk. Fault-tolerant behavior in state-of-the-art grid workflow management systems. 2007.

[6] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[7] Pedro Mejía-Alvarez and Daniel Mossé. A responsiveness approach for scheduling fault recovery in real-time systems. In *Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE*, pages 4–13. IEEE, 1999.

[8] K. Plankensteiner and R. Prodan. Meeting soft deadlines in scientific workflows using resubmission impact. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):890–901, May 2012.

[9] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2016.

[10] Rodrigo N Calheiros and Rajkumar Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1787–1796, 2014.

[11] Doaa M Abdelkader and Fatma Omara. Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. *Egyptian Informatics Journal*, 13(2):135–145, 2012.

[12] Ji Wang, Weidong Bao, Xiaomin Zhu, Laurence T Yang, and Yang Xiang. Festal: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds. *IEEE Transactions on Computers*, 64(9):2545–2558, 2015.

[13] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[14] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.